The Missing GAP: From Solving Square Jigsaw Puzzles to Handling Real World Archaeological Fragments

Supplementary Material

7. Introduction

This supplementary material provides comprehensive technical documentation for all components of our work. We organize the content into three main sections: (1) the GAP dataset generation pipeline and statistical validation (Section 8), (2) complete implementation details and training configurations for PuzzleFlow and all baseline methods (Section 9), and (3) qualitative results showcasing puzzle reconstructions (Section 11). Together, these sections enable full reproduction of our experiments and provide deeper insights into our methodological choices.

8. GAP Dataset: Generation and Validation

This section details the complete pipeline for generating the GAP (Generated Archaeological-fragments Puzzles) datasets, including the fragment generator architecture, training procedure, source data collection, and comprehensive statistical validation against real archaeological fragments.

8.1. Fragment Generator Architecture

Our fragment generator employs a Variational Autoencoder (VAE) [23] trained on binary mask representations of 958 real archaeological fragments from the RePAIR dataset [55]. The architecture is visualized in Figure 6.

8.1.1. Encoder Architecture

The encoder compresses 128×128 binary fragment masks into a 64-dimensional latent representation:

- **Conv1**: $128 \times 128 \times 1 \rightarrow 64 \times 64 \times 32$
  - 3×3 kernel, stride 2, padding 1
  - ReLU activation, BatchNorm, Dropout(0.3)
- **Conv2**: $64 \times 64 \times 32 \rightarrow 32 \times 32 \times 64$
  - 3×3 kernel, stride 2, padding 1
  - ReLU activation, BatchNorm, Dropout(0.3)
- **Conv3**: $32 \times 32 \times 64 \rightarrow 16 \times 16 \times 128$
  - 3×3 kernel, stride 2, padding 1
  - ReLU activation, BatchNorm, Dropout(0.3)
- **Conv4**: $16 \times 16 \times 128 \rightarrow 8 \times 8 \times 256$
  - 3×3 kernel, stride 2, padding 1
  - ReLU activation, BatchNorm, Dropout(0.3)
- **Latent projection**: Flatten to 16,384-D, then project to 64-D μ and 64-D $\log \sigma^2$

8.1.2. Decoder Architecture

The decoder reconstructs fragment masks from 64-dimensional latent codes:

- **Latent expansion**: 64-D $\rightarrow 8 \times 8 \times 256$ (reshape)
- **TransConv1**: $8 \times 8 \times 256 \rightarrow 16 \times 16 \times 128$
  - 3×3 kernel, stride 2, padding 1, output_padding 1
  - ReLU activation, BatchNorm
- **TransConv2**: $16 \times 16 \times 128 \rightarrow 32 \times 32 \times 64$
  - 3×3 kernel, stride 2, padding 1, output_padding 1
  - ReLU activation, BatchNorm
- **TransConv3**: $32 \times 32 \times 64 \rightarrow 64 \times 64 \times 32$
  - 3×3 kernel, stride 2, padding 1, output_padding 1
  - ReLU activation, BatchNorm
- **TransConv4**: $64 \times 64 \times 32 \rightarrow 128 \times 128 \times 1$
  - 3×3 kernel, stride 2, padding 1, output_padding 1
  - Sigmoid activation (output in [0,1])

8.1.3. Training Configuration

- **Loss function**: $\mathcal{L} = \text{BCE}(x, \hat{x}) + \beta \cdot D_{\text{KL}}(q(z|x)\|\mathcal{N}(0, I))$ where $\beta = 1.0$
- **Optimizer**: Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
- **Learning rate**: 10^{-4} (constant, no scheduling)
- **Batch size**: 32
- **Epochs**: 44 (early stopping based on validation loss)
- **Best validation loss**: 1623.76 (achieved at epoch 44)
- **Training data**: 958 binary masks from RePAIR dataset (80/10/10 train/val/test split)
- **Hardware**: NVIDIA RTX 4070 GPU (8GB VRAM).

8.2. Source Image Collection

We utilize artwork images from The Metropolitan Museum of Art's Open Access collection [36], accessed via their public API. The collection process ensures high-quality, diverse cultural heritage imagery suitable for synthetic archaeological puzzle generation.

8.2.1. Collection Pipeline

1. **API Query**: Query `collectionapi.metmuseum.org` for public domain objects
2. **Filtering**: Apply `isPublicDomain=True` AND `title NOT LIKE '%fragment%'`, in order to assure selected images are indeed categorized as public domain, while filtering out images of already fragmented artifacts.
3. **Sampling**: Random selection of 40,000 unique object IDs (20,000 for GAP-3, 20,000 for GAP-5)
4. **Download**: Parallel retrieval with 20 workers and retry logic for failed requests
5. **Storage**: Full-resolution primary images with complete metadata

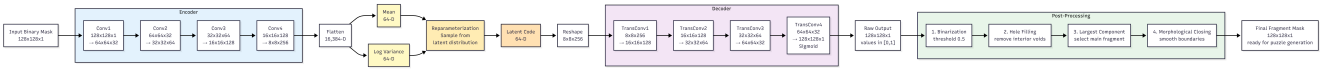

Figure 6. **Fragment Generator Architecture.** Our VAE encodes 128×128 binary fragment masks through four convolutional layers into a 64-dimensional latent space, then reconstructs synthetic fragments via transposed convolutions. The reparameterization trick enables sampling diverse fragments during training while maintaining archaeological realism.

6. **Metadata**: CSV files with object ID, title, artist information, date/period, department, culture, medium, and dimensions (where available in the MET's original metadata)

8.2.2. Collection Diversity Statistics

Analysis of the 40,000 collected images reveals exceptional temporal, geographical, and medium diversity:

Departmental Distribution:
- 19 unique departments represented
- Top 5: Drawings & Prints (29.8%), European Sculpture & Decorative Arts (14.4%), Asian Art (13.8%), Greek & Roman Art (5.7%), Egyptian Art (5.4%)

Temporal Coverage:
- Range: 970 BCE to 2000 CE (∼2,970 years)
- Distribution: 19th century (24.6%), 16th-17th centuries (11.7%), 18th century (11.5%), ancient-medieval periods (3.8%), before 0 CE (2.6%)

Media Representation:
- Prints (21.6%), metalwork (15.4%), textiles (10.4%), ceramics (10.2%), drawings (8.3%), photographs (5.6%), sculptures (3.6%), paintings (2.1%)

Cultural Origins:
- 1,933 unique cultures represented
- Top 5: Japanese (12.9%), American (10.2%), Chinese (8.1%), French (7.0%), Italian (3.0%)

Dataset Separation: GAP-3 and GAP-5 use completely disjoint sets of 20,000 images each, ensuring independent evaluation without image overlap.

8.3. Statistical Validation

We validate that VAE-generated fragments preserve the statistical distribution of real archaeological fragments through comprehensive shape analysis.

8.3.1. Geometric Feature Definitions

We formally define the eight geometric features extracted from fragment binary masks $M \in \{0,1\}^{H \times W}$:

1. **Area** $A = \sum_{i,j} M(i,j)$: Total number of foreground pixels, providing an absolute size measure in px^2.
2. **Perimeter** P: Length of the fragment boundary computed via contour tracing, measured in pixels. Captures edge extent and complexity.
3. **Aspect Ratio** $r = w_{\text{bbox}}/h_{\text{bbox}}$: Ratio of minimum bounding rectangle width to height. Note that original fragments were normalized to square bounding boxes (aspect ratio ≈ 1) pre-training to ensure consistent input dimensions (128×128 pixels), resulting in distributions centered near unity.
4. **Solidity** $S = A/A_{\text{hull}}$: Ratio of fragment area to its convex hull area. $S = 1$ for convex fragments; $S < 1$ quantifies boundary concavity depth. Formally, $A_{\text{hull}} = \text{Area}(\text{ConvexHull}(M))$.
5. **Circularity** $C = 4\pi A/P^2$: Isoperimetric quotient comparing shape to a circle. $C = 1$ for perfect circles; $C < 1$ for irregular shapes. Invariant to scaling.
6. **Compactness** $K = P^2/A$: Inverse measure of shape efficiency. Lower values indicate more compact shapes; higher values reflect irregular boundaries. Related to circularity by $K = 4\pi/C$.
7. **Vertices** V: Number of vertices in the convex hull approximation, computed via Douglas-Peucker algorithm with tolerance $\epsilon = 0.01P$. Represents corner count and polygon complexity.
8. **Concavities** N: Number of contour points exhibiting negative curvature (inward bending), computed via discrete derivative approximation: $N = |\{p \in \partial M : \kappa(p) < -\tau\}|$ where $\kappa(p)$ is local curvature and τ is a small threshold. Quantifies edge irregularity.

These features capture complementary aspects of fragment morphology: global size (area), boundary characteristics (perimeter, circularity, compactness), shape regularity (solidity, aspect ratio), and fine-scale structure (vertices, concavities).

8.3.2. Summary Statistics

Table 4 presents comprehensive statistics comparing real and synthetic fragments across all features.

8.3.3. Box Plot Visualizations

Figure 7 presents box plots showing medians, interquartile ranges, and outliers for all features, demonstrating the close alignment between real and synthetic fragment distributions.

8.3.4. Dimensionality Reduction Analysis

Principal Component Analysis (PCA) on the 8-dimensional feature space reveals:

Table 4. Detailed summary statistics comparing real (RePAIR) and synthetic (VAE-generated) fragments across eight geometric features (N=958 each).

Feature	Group	Mean ± SD	Median	Min–Max	IQR
Area (px^2)	Real	10,617 ± 1,333	10,598	7,245–14,821	1,746
	Synthetic	10,716 ± 863	10,705	8,012–13,254	1,142
Perimeter (px)	Real	451.95 ± 29.72	449.80	370.2–542.6	39.8
	Synthetic	434.88 ± 27.60	433.10	361.7–518.4	36.2
Aspect Ratio	Real	1.001 ± 0.023	0.998	0.945–1.089	0.031
	Synthetic	0.996 ± 0.055	0.993	0.842–1.178	0.072
Solidity	Real	0.931 ± 0.027	0.935	0.845–0.986	0.036
	Synthetic	0.936 ± 0.030	0.941	0.831–0.991	0.040
Circularity	Real	0.655 ± 0.068	0.661	0.471–0.812	0.089
	Synthetic	0.718 ± 0.081	0.726	0.504–0.895	0.107
Compactness	Real	19.47 ± 2.90	19.23	13.12–28.61	3.81
	Synthetic	17.78 ± 2.41	17.53	11.92–25.42	3.16
Vertices	Real	9.73 ± 2.12	10	5–16	3
	Synthetic	11.78 ± 2.03	12	6–18	3
Concavities	Real	19.79 ± 3.62	20	11–31	5
	Synthetic	23.01 ± 3.74	23	13–35	5

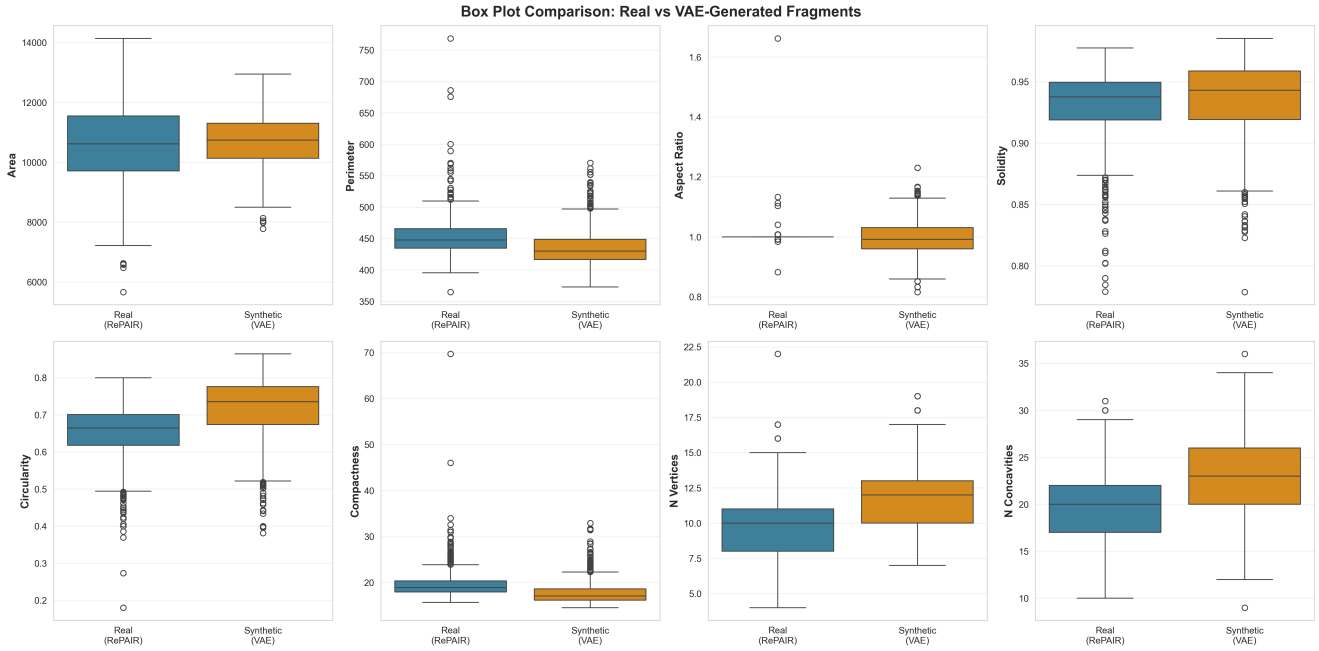

Figure 7. **Distribution comparison via box plots.** Real (RePAIR) fragments shown in blue, synthetic (VAE) fragments in orange. Boxes indicate interquartile ranges (IQR), horizontal lines show medians, whiskers extend to 1.5×IQR, and circles represent outliers. Core shape properties (area, solidity) exhibit high similarity, while edge complexity metrics show expected smoothing effects from VAE reconstruction.

- **PC1 (45.4% variance)**: Overall fragment size and complexity (high loadings: area, perimeter, vertices, concavities)
- **PC2 (17.8% variance)**: Edge irregularity and compactness (high loadings: circularity, compactness)
- **PC3 (13.1% variance)**: Aspect ratio and orientation
- **PC4–PC8 (23.7% variance)**: Higher-order shape variations

The first two principal components capture 63.2% of total variance. As shown in Figure 5 (main paper), real and synthetic fragments exhibit substantial overlap in this reduced space, with no isolated clusters or systematic biases, confirming that the VAE successfully captures the underlying distribution of archaeological fragment shapes.

These validation results confirm that our VAE successfully captures the geometric essence of archaeological fragments while maintaining practical advantages for large-scale dataset generation. High fidelity in core shape features (1-3% differences in area and solidity) and substantial PCA overlap demonstrate that GAP fragments authentically represent real artifact morphology. Expected smoothing in edge complexity metrics reflects VAE reconstruction characteristics but preserves the irregular, non-linear erosion patterns absent from existing square-piece datasets. This combination of archaeological realism and synthetic scalability positions GAP as an effective bridge between simplified academic benchmarks and real-world heritage reconstruction, enabling systematic algorithm development on challenging, realistic fragment geometries at a scale impossible with limited authentic artifact collections.

9. Implementation Details: Models and Baselines

This section provides complete implementation details for PuzzleFlow and all baseline methods, enabling full reproducibility of our experimental results.

9.1. PuzzleFlow: Architecture and Training

PuzzleFlow combines a pretrained Vision Transformer backbone with additional transformer layers and discrete flow matching for puzzle reassembly. The architecture is visualized in Figure 8.

9.1.1. Architecture Specifications

Table 5 details the complete PuzzleFlow architecture.

9.1.2. Training Hyperparameters

All training was conducted on NVIDIA RTX 4090 GPUs with 24GB memory. Table 6 provides complete training configuration.

9.1.3. Loss Function and Training Objective

The training objective combines cross-entropy loss over predicted positions at randomly sampled timesteps $t \in [0, 1]$ during the flow process:

$$\mathcal{L} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\mathbf{x}_0,\mathbf{x}_1}\left[-\sum_{i=1}^{N}\log p_\theta(x_1^{(i)}|\mathbf{x}_t, t)\right] \quad (7)$$

where $\mathbf{x}_0$ represents the initial scrambled permutation, $\mathbf{x}_1$ is the target solved configuration, $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}_1$ is the linearly interpolated state, and N is the number of fragments.

Table 5. PuzzleFlow architecture specifications. Configuration is identical across GAP-3 and GAP-5 datasets.

Component	Configuration
Visual Encoder	
Backbone	ViT-Base-Patch16-224 [11]
Initialization	Pretrained on ImageNet-21k
Hidden dimension	768
Attention heads (ViT)	12
Encoder layers (ViT)	12
Patch size	16×16 pixels
Input resolution	224×224 per fragment
Gradient checkpointing	Enabled
Transformer Decoder	
Hidden dimension (d_{model})	768
Attention heads	12
Decoder layers	4
Feedforward dimension	3072 ($4\times d_{\text{model}}$)
Dropout rate	0.1
Flow Matching	
Interpolation type	Linear
Time embedding	Sinusoidal (dim=768)
Position embedding	2-layer MLP ($2 \to 256 \to 128$)
Output head	3-layer MLP ($768 \to 512 \to 256 \to N$)
Dataset-Specific Parameters	
GAP-3 fragment size	128×128 pixels
GAP-5 fragment size	128×128 pixels
Solved image size (GAP-3)	384×384 pixels (3×128)
Solved image size (GAP-5)	640×640 pixels (5×128)
Grid size (GAP-3)	3×3 (9 positions)
Grid size (GAP-5)	5×5 (25 positions)

9.1.4. Implementation Optimizations

Several techniques were utilized for efficient training:

- **Gradient checkpointing**: Reduces memory usage by ∼30% by recomputing activations during the backward pass rather than storing them.
- **Mixed precision training**: Automatic Mixed Precision (AMP) with FP16 enabled, providing 1.5–2× speedup and 40% memory reduction while maintaining numerical stability through automatic loss scaling.
- **Adaptive batch sizing**: Training uses batch size 8, but validation uses batch size 2 to prevent out-of-memory errors during the multi-step sampling process.
- **Fast validation**: During training, validation uses 5 flow steps for speed; final evaluation uses 20 steps.

9.2. Learning-Based Baseline Adaptations

We evaluate three state-of-the-art learning-based methods: FCViT [22], JPDVT [29], and PuzLM [12]. Since FCViT and JPDVT were originally designed for square RGB puzzles with internal shuffling mechanisms, we reconstructed solved puzzle images by placing GAP's RGBA fragments (with

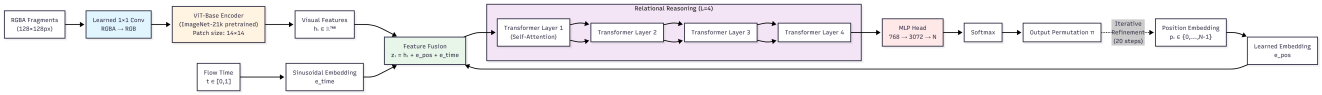

Figure 8. **PuzzleFlow Architecture.** Individual puzzle fragments are processed through a pretrained ViT backbone to extract 768-dimensional visual features. These features are combined with position embeddings (encoding current fragment placements) and time embeddings (encoding flow matching timestep), then passed through 4 additional transformer layers for cross-piece reasoning. The output head predicts logits over all possible positions for each fragment. During training, we sample random timesteps $t \in [0, 1]$ and interpolate between scrambled and solved states. During inference, we iteratively denoise from random initialization to the solved configuration.

Table 6. PuzzleFlow training hyperparameters. Configuration is consistent across both GAP-3 and GAP-5 datasets.

Parameter	Value
Optimization	
Optimizer	AdamW [30]
Learning rate	1×10^{-5}
Weight decay	0.01
Betas	(0.9, 0.999)
Learning rate schedule	OneCycleLR
Warmup percentage	10%
Gradient clipping	None
Training Configuration	
Number of epochs	30
Batch size (training)	8
Batch size (validation)	2
Mixed precision (AMP)	Enabled (FP16)
Validation flow steps	5 (20 for final eval)
Data Loading	
Number of workers	8
Persistent workers	Enabled
Prefetch factor	2
Pin memory	Enabled
Regularization	
ViT backbone	Fine-tuned (unfrozen)
Frozen ViT layers	0
Dropout	0.1

alpha channel dropped) at their ground truth grid positions, resized to method-specific dimensions. PuzLM operates on individual scrambled fragments and required only conversion from RGBA to RGB format. All methods were trained for 30 epochs using default hyperparameters from their official repositories. For JPDVT, we used JPDVT-T variant.

Note on Preprocessing Rationale. This preprocessing step is necessary because GAP fragments are provided as individual RGBA images with irregular shapes, whereas the baseline methods expect complete grid-aligned images that they internally shuffle during training. By reconstructing solved puzzles and allowing each method to perform its own internal shuffling, we ensure fair comparison under each method's original design assumptions.

9.3. Classical Baseline Implementations

9.3.1. Greedy Solver (Pomeranz et al. 2011)

We implemented the fully automated greedy solver of Pomeranz *et al.* [39], which constructs puzzles through iterative best-buddy placement.

Compatibility Metric. We compute pairwise dissimilarity in LAB color space. For adjacent pieces x_i and x_j, the dissimilarity along edge direction r is:

$$D(x_i, x_j, r) = \sum_k \left(|2e_i^k - e_i^{k-1} - e_j^k|^P + |2e_j^k - e_j^{k+1} - e_i^k|^P \right)^{Q/P} \tag{8}$$

where e_i^k denotes the k-th pixel along the edge of piece x_i, $r \in \{\text{LEFT, RIGHT, UP, DOWN}\}$, and $P = 0.3$, $Q = 0.0625$ are constants from [39].

Compatibility is computed as:

$$C(x_i, x_j, r) = \exp\left(-\frac{D(x_i, x_j, r)}{\text{percentile}_{25}(D(x_i, \cdot, r))} \right) \tag{9}$$

Best Buddy Definition. Pieces x_i and x_j are *best buddies* in direction r if:

$$\arg\max_k C(x_i, k, r) = j \quad \text{and} \quad \arg\max_k C(x_j, k, \bar{r}) = i \tag{10}$$

where $\bar{r}$ denotes the opposite direction.

Algorithm. The solver proceeds in three phases:

1. **Seed selection**: Choose the piece with maximum mutual best buddies as initial seed, placed at grid center.
2. **Greedy placement**: Iteratively select candidate slots (positions adjacent to placed pieces with maximum occupied neighbors) and assign pieces with highest average compatibility. Ties are broken using best-buddy relationships.
3. **Refinement**: Segment assembly into connected components based on best-buddy relationships. Keep only the largest segment, re-center, and repeat until no improvement in the best-buddies metric (BBM).

The best-buddies metric evaluates solution quality:

$$\text{BBM} = \frac{\text{\# best-buddy edges}}{\text{\# total edges}} \tag{11}$$

Implementation Details.

- Input: Fragment images converted to LAB color space using `scikit-image`
- Precomputation: Full $4 \times N \times N$ dissimilarity matrix for all directions
- Grid handling: Dynamic expansion via NumPy array rolling when boundaries are reached
- Termination: Algorithm stops when BBM no longer improves

Computational Complexity. Building the dissimilarity matrix requires $O(N^2 \cdot H)$ operations for N pieces of height H pixels. The placement phase is $O(N \cdot k)$ where k is the number of iterations (typically 3–5). Runtime: 1–2 minutes for 3×3 puzzles, 5–10 minutes for 5×5 puzzles on a single CPU core.

9.3.2. Genetic Algorithm Solver (Sholomon et al. 2013)

We implemented the genetic algorithm approach of Sholomon *et al.* [46], which frames puzzle solving as permutation optimization.

Representation. Each individual is a permutation $\pi \in S_N$ of piece indices, where position i contains piece $\pi(i)$.

Fitness Function. Fitness is the negative sum of dissimilarities between adjacent pieces:

$$f(\pi) = - \sum_{(i,j) \text{ adjacent}} D(\pi(i), \pi(j), r_{i \to j}) \qquad (12)$$

Higher fitness (lower total dissimilarity) indicates better solutions.

Genetic Operators.

- **Selection**: Tournament selection with tournament size 3
- **Crossover**: Partially Mapped Crossover (PMX) with rate 0.8
- **Mutation**: Three types (swap, inversion, scramble) with rate 0.01
- **Elitism**: Retain top 10% unchanged

Algorithm Parameters.

- Population size: 100
- Maximum generations: 1000
- Early stopping: 100 generations without improvement
- Mutation rate: 0.01
- Crossover rate: 0.8
- Elitism ratio: 0.1

Computational Complexity. Each generation requires $O(P \cdot G^2)$ fitness evaluations for population size P and grid size G. Total complexity is $O(T \cdot P \cdot G^2)$ for T generations. Runtime: 2–5 minutes for 3×3 puzzles, 15–30 minutes for 5×5 puzzles on a single CPU core.

9.3.3. Adaptation to GAP Datasets

Both classical methods were adapted to handle GAP's irregular fragments:

- **Color space conversion**: RGBA → RGB → LAB using `scikit-image`
- **Edge handling**: Dissimilarity computed along detected fragment boundaries (non-zero alpha channel regions)
- **Erosion robustness**: No special handling for erosion; methods rely purely on boundary compatibility, which degrades as erosion increases

The primary limitation of these classical approaches on GAP is their reliance on edge continuity. As erosion removes original boundaries, the compatibility metrics become less informative, leading to degraded performance compared to learning-based methods that leverage global visual patterns.

9.4. Evaluation Protocol

All methods are evaluated using consistent metrics on held-out test sets:

- **Exact Match Rate (Perfect Accuracy)**: Percentage of puzzles with all pieces correctly placed.
- **Position Accuracy (Direct Accuracy)**: Average fraction of correctly placed pieces per puzzle.
- **Spatial Relationship Accuracy (SRA)**: Average fraction of correctly adjacent piece pairs, as defined in the main paper.

For PuzzleFlow and JPDVT, we use 20-step sampling during evaluation. Classical methods produce deterministic outputs.

10. Validation of PuzzleFlow on simpler settings

Although PuzzleFlow is introduced mainly as a strong reference solver rather than the core focus of this work, have trained and evaluated this framework on the widely used JPwLEG-3 and JPwLEG-5 datasets, without any dataset-specific tuning, achieving absolute accuracy (AA) of 0.726 and perfect accuracy (PA) of 0.437 on JPwLEG-3; and AA of 0.290 and PA of 0 on JPwLEG-5. While these results do not outperform the SOTA on this benchmark, they are competitive with, and in some cases surpass, recent prominent approaches such as JPDVT, despite being obtained by a solver *not specialized* to square-piece setting.

Importantly, the performance is consistent with that observed on our corresponding GAP datasets, supporting that the proposed permutation-flow formulation is robust across both irregular and conventional square-piece puzzles rather than overfitting to GAP alone. See Table. 7 for full results.

Table 7. Validation on square-piece JPwLEG benchmarks. Some approaches did not report performance in all metrics, or in both dataset variations

Method	**JPwLEG-3** (3×3)		**JPwLEG-5** (5×5)	
	AA (%)	**PA (%)**	**AA (%)**	**PA (%)**
JPDVT [29]	71.3	N/A	N/A	N/A
Greedy [39]	71.5	40.0	24.1	0.1
Deepzzle [38]	74.0	44.9	21.9	0.0
Tabu [1]	73.8	44.8	24.6	0.0
GA [46]	73.9	44.9	25.1	0.0
PuzzleFlow (ours)	72.6	43.7	29.0	0.0
SD^{2}RL [49]	81.6	59.7	40.3	5.1
FCViT [22]	**96.9**	**87.9**	N/A	N/A
PDN-GA [50]	81.3	58.2	44.3	6.1
ERL-MPP [52]	N/A	N/A	52.7	18.6
VLHSA [63]	85.4	N/A	<u>66.9</u>	<u>19.0</u>
PuzLM [12]	<u>91.9</u>	<u>84.5</u>	**72.1**	**32.5**

11. Qualitative Results

Figure 9 shows representative examples of PuzzleFlow solving GAP-3 and GAP-5 puzzles, including both successful reconstructions and challenging failure cases. Note that in some cases generally low evaluation scores could be explained by relatively similar pieces, mistakenly assigned to the correspondent position.

11.1. Ethical Statement

Our research uses only publicly available artifact images and metadata from the MET collection (CC0 license), ensuring compliance with data ownership and privacy regulations. The fragments in our newly presented GAP dataset are synthetic, generated without any human-derived personal information or sensitive content. We believe the potential for misuse is minimal; however, we acknowledge that automatic assembly tools could theoretically be misapplied to heritage items without proper authority. We encourage responsible use strictly within permitted conservation, restoration, and academic boundaries. All datasets and code will be released according to museum guidelines and community standards.

11.2. Code and Data Availability

Complete implementation code and the GAP datasets have been publically released. The codebase includes: The complete GAP datasets, along with the generation code (VAE training, fragment synthesis) and trained VAE checkpoint, PuzzleFlow training and inference scripts, and some adapted baseline implementations.

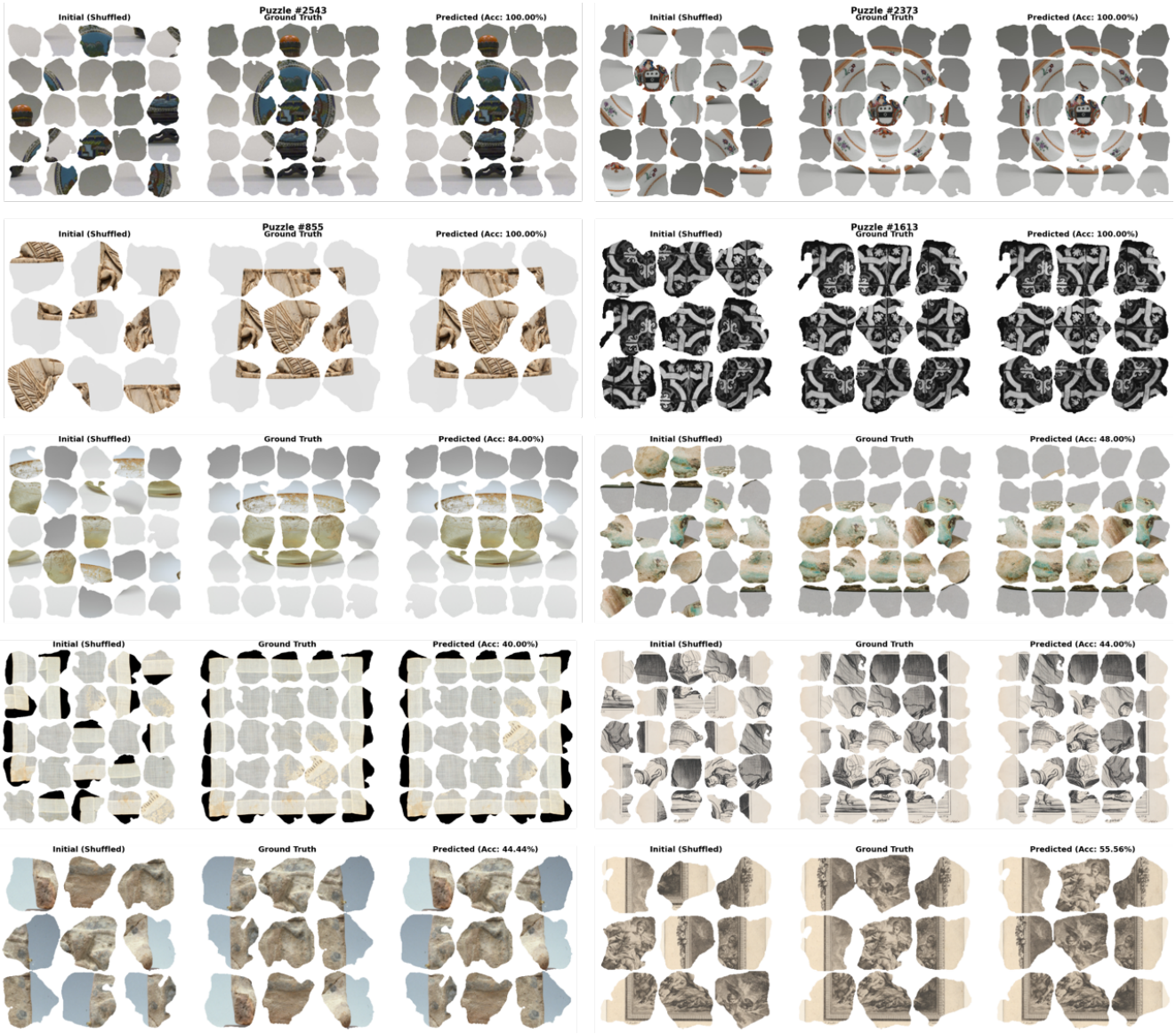

Figure 9. **Qualitative Results.** Representative examples of PuzzleFlow solving GAP puzzles. **Top rows:** Successful reconstructions on GAP-3 (left) and GAP-5 (right) with heavily eroded fragments. **Bottom rows:** Challenging failure cases where erosion or visual ambiguity leads to errors. PuzzleFlow successfully handles irregular fragment geometries and leverages global visual patterns, though some puzzles with extreme erosion or repetitive textures remain challenging.